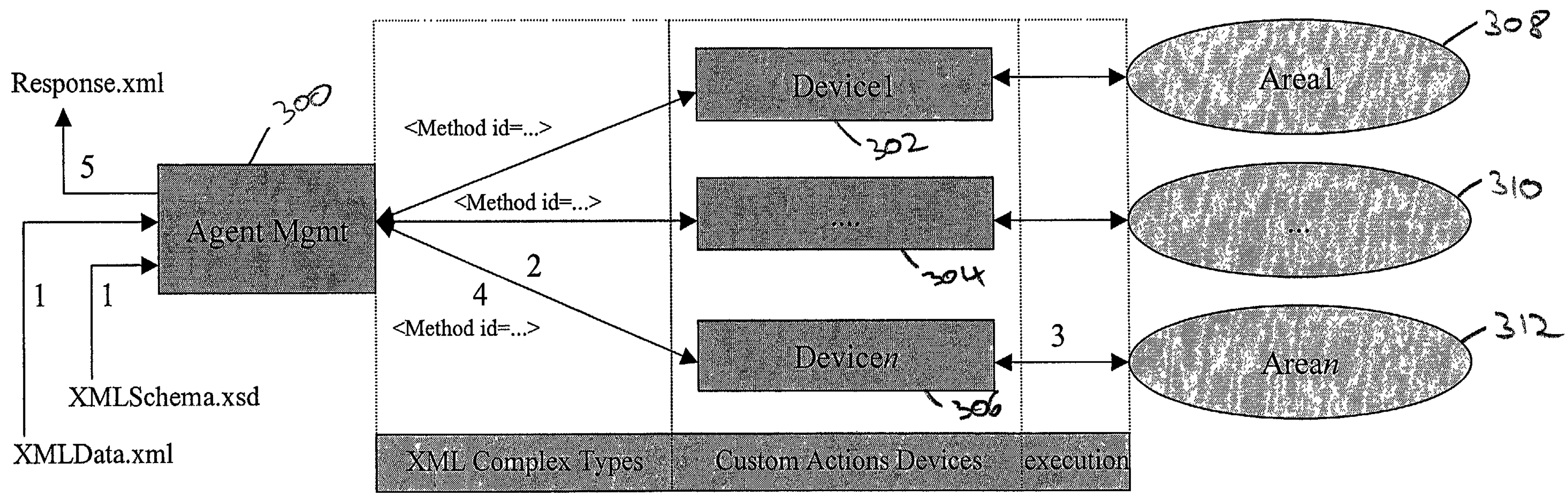
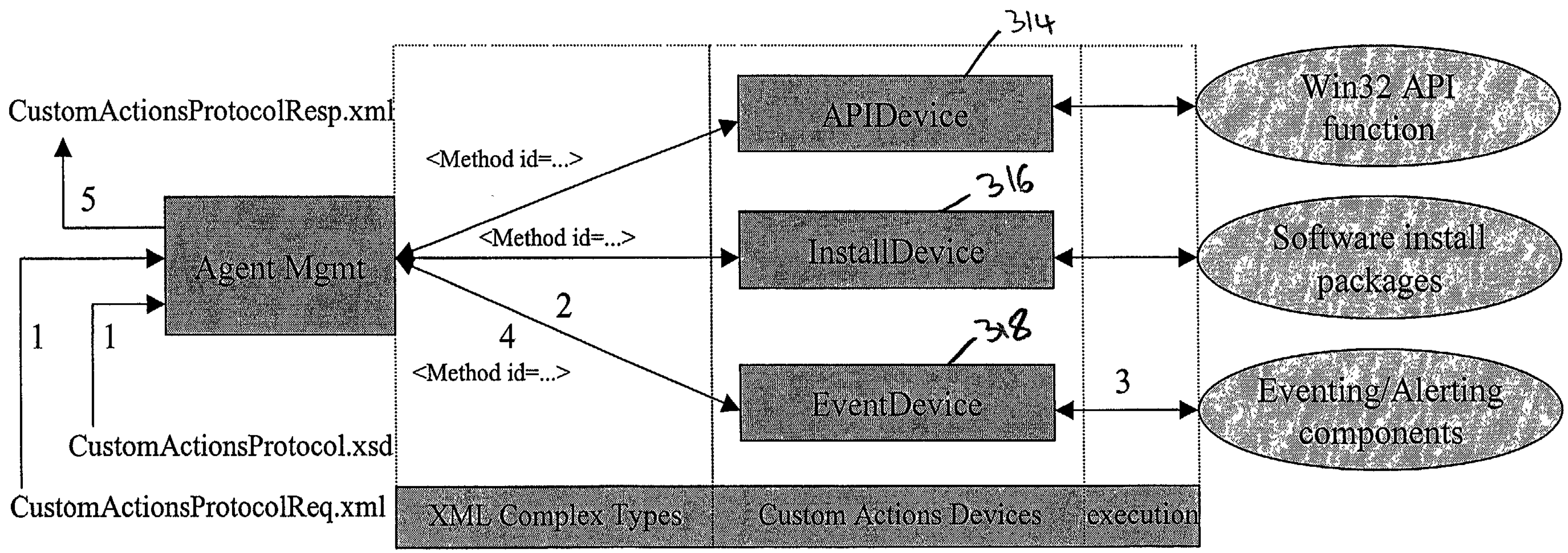
Architecture overview



1) Request XML file is passed to Agent Mgmt containing complex types
   Agent Mgmt splits the data into parts and hands the <Method> over to the corresponding Custom Actions Devices
2) The complex type <Method> describes the method to execute and its parameters
3) Custom Actions Devices execute the functions in a certain area they are responsible for and pack the result into XML complex types
4) Complex types containing result data are returned to Agent Mgmt
5) XML stream has been packed and is returned

Fig. 1

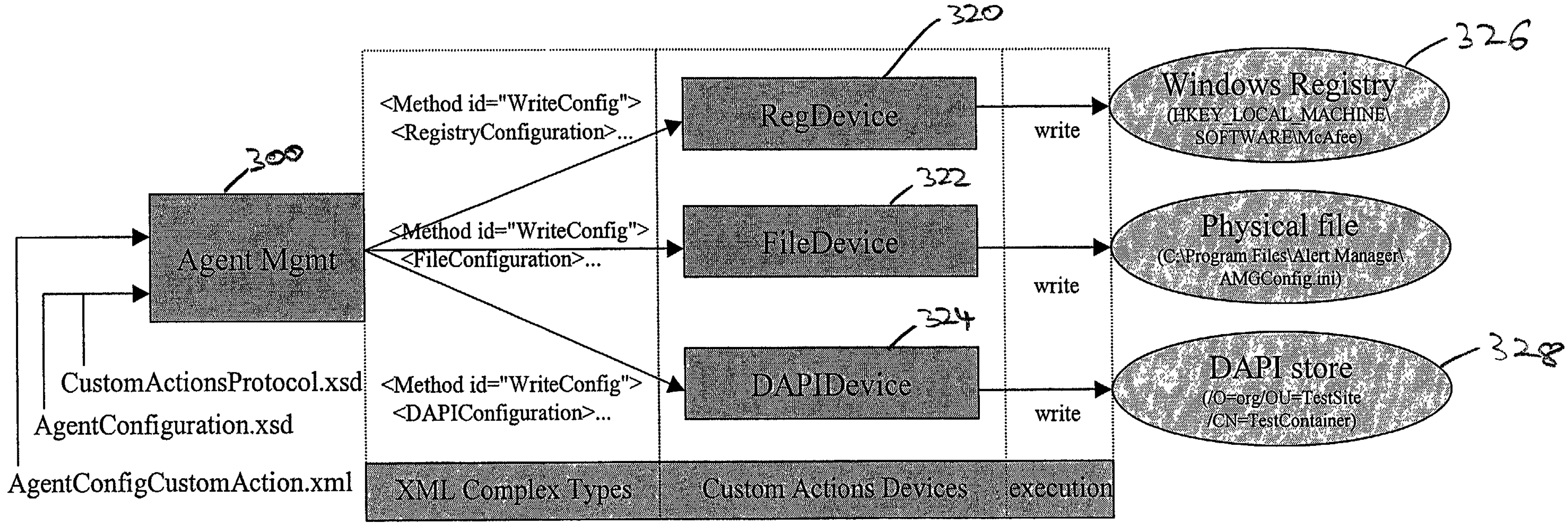
Executing Custom Actions



1) Request XML file is passed to Agent Mgmt containing complex types
2) The complex type <Method> describes the method to execute and its parameters
   Each <Method> is passed to the corresponding Custom Actions Device
3) Custom Actions Devices execute the functions and pack the result into XML complex types
4) Complex types containing result data are returned to Agent Mgmt
5) XML stream has been packed and is returned

Fig. 2

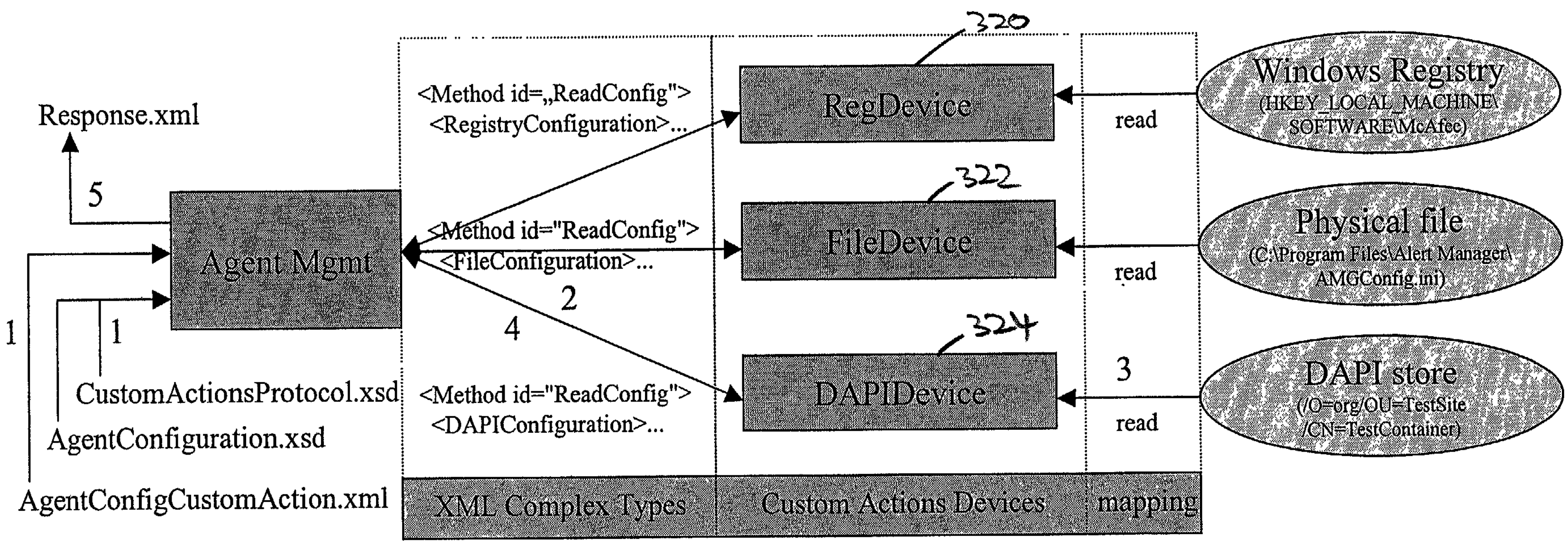
Deploying Configuration Data



1) Agent Mgmt receives AgentConfigCustomAction.xml
2) .xml file is validated against .xsd file(s) to make sure valid data is written
3) XML Complex types are sent to Custom Actions Devices, the parameters containing the configuration data
4) Custom Actions Devices update the config store they are responsible for
5) Optionally a return value can be returned as a Response.xml file

Fig. 3

Retrieving Configuration Data



1) Request XML file is passed to Agent Mgmt containing empty requested complex types
2) The empty complex types are passed to the Custom Actions Device
3) Complex types are filled by mapping required data from store
4) Complex types containing data are returned to Agent Mgmt
5) XML stream has been packed and is returned

Fig. 4

Initiator

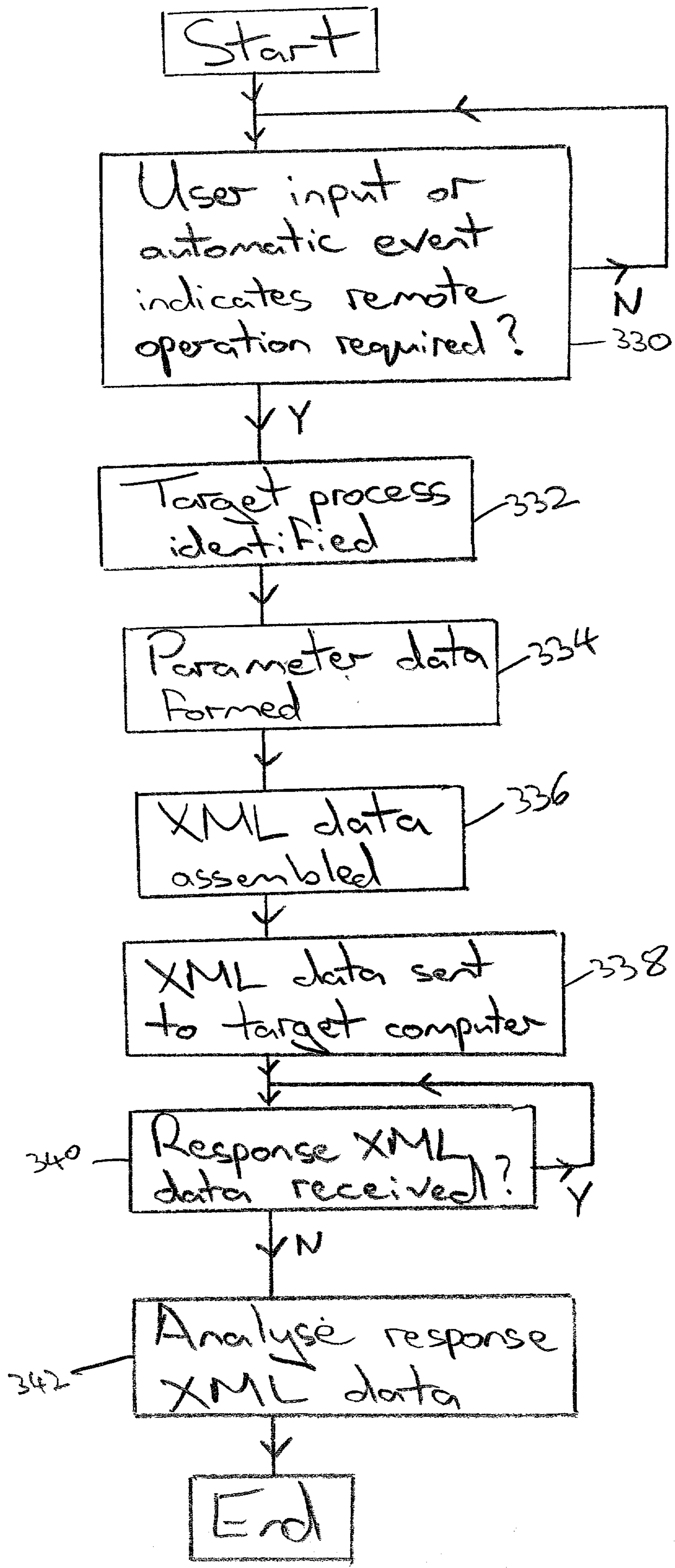


Fig. 5

Agent

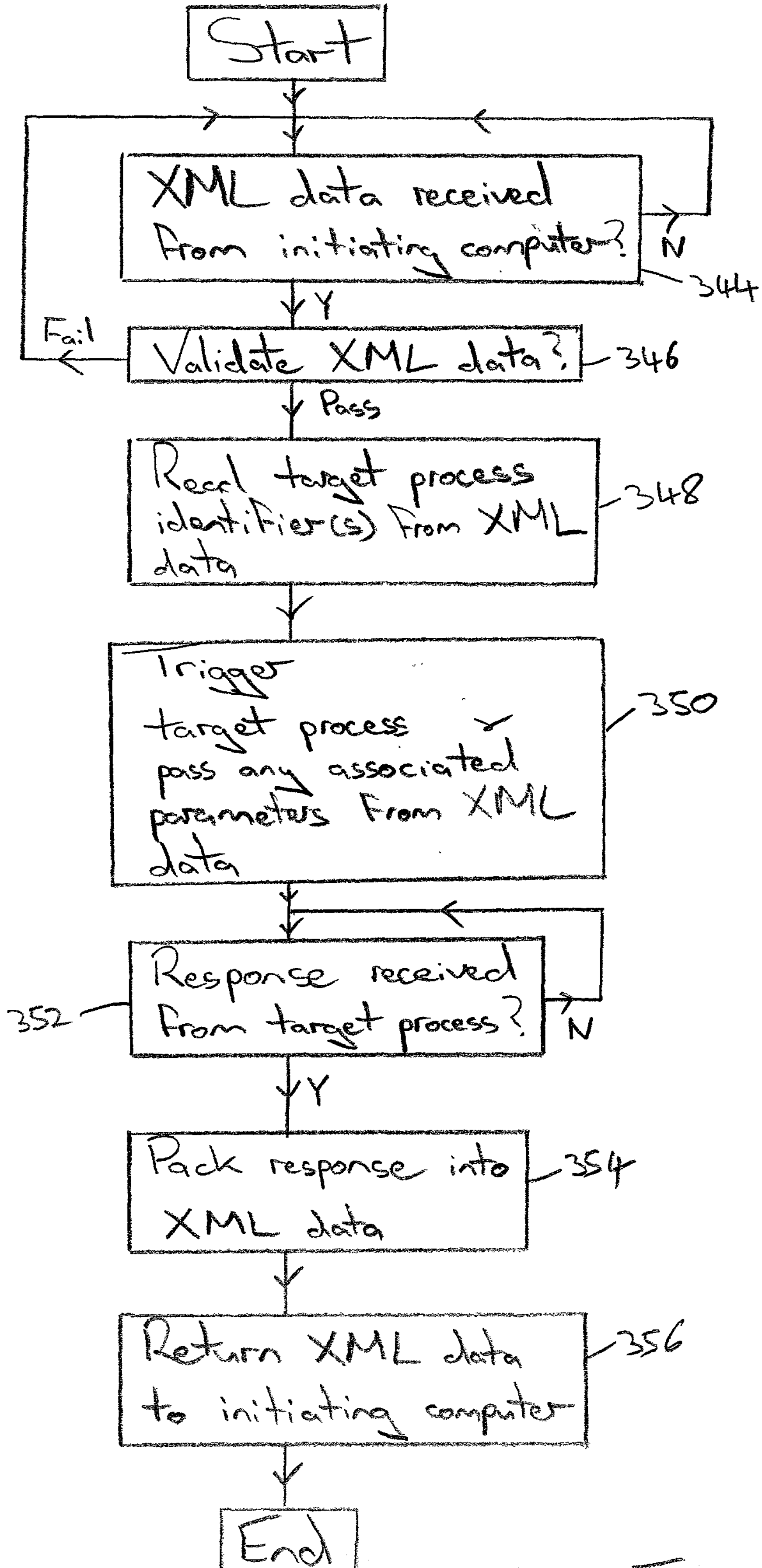

Start
XML data received from initiating computer?
N
344
Y
Fail
Validate XML data?
346
Pass
Read target process identifier(s) from XML data
348
Trigger target process & pass any associated parameters from XML data
350
Response received from target process?
352
N
Y
Pack response into XML data
354
Return XML data to initiating computer
356
End

Fig. 6

Target

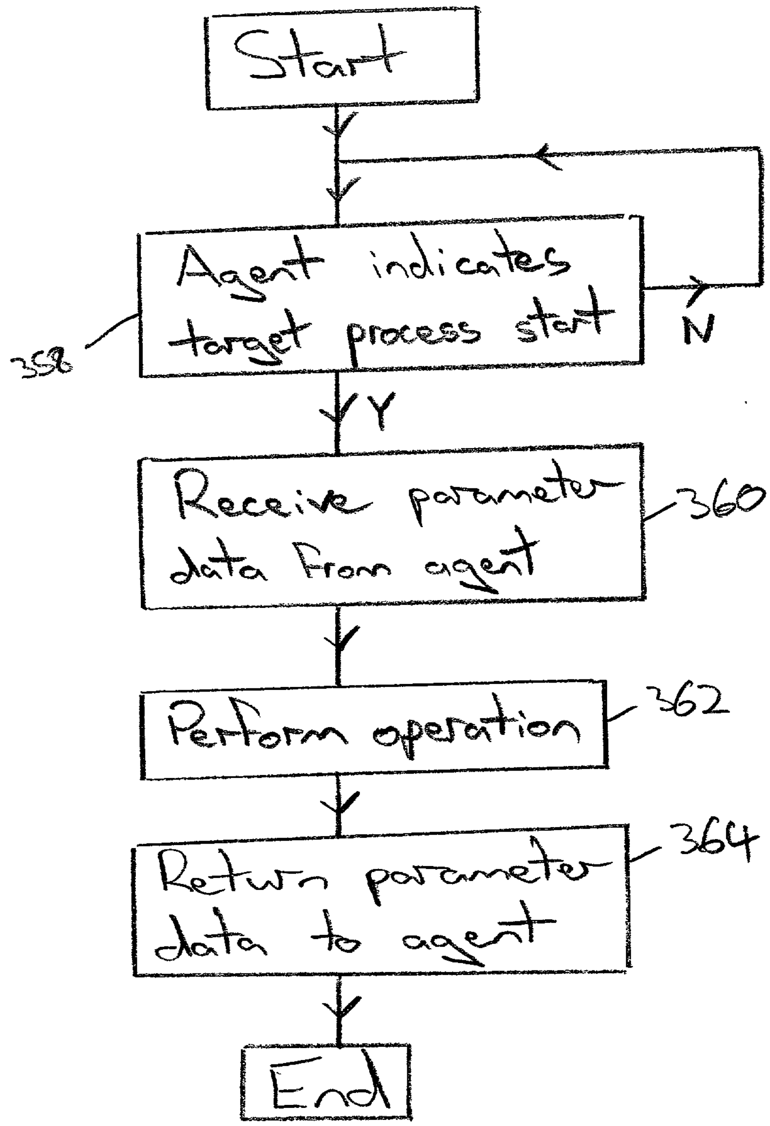

Start
Agent indicates target process start
N
Y
358
Receive parameter data from agent
360
Perform operation
362
Return parameter data to agent
364
End

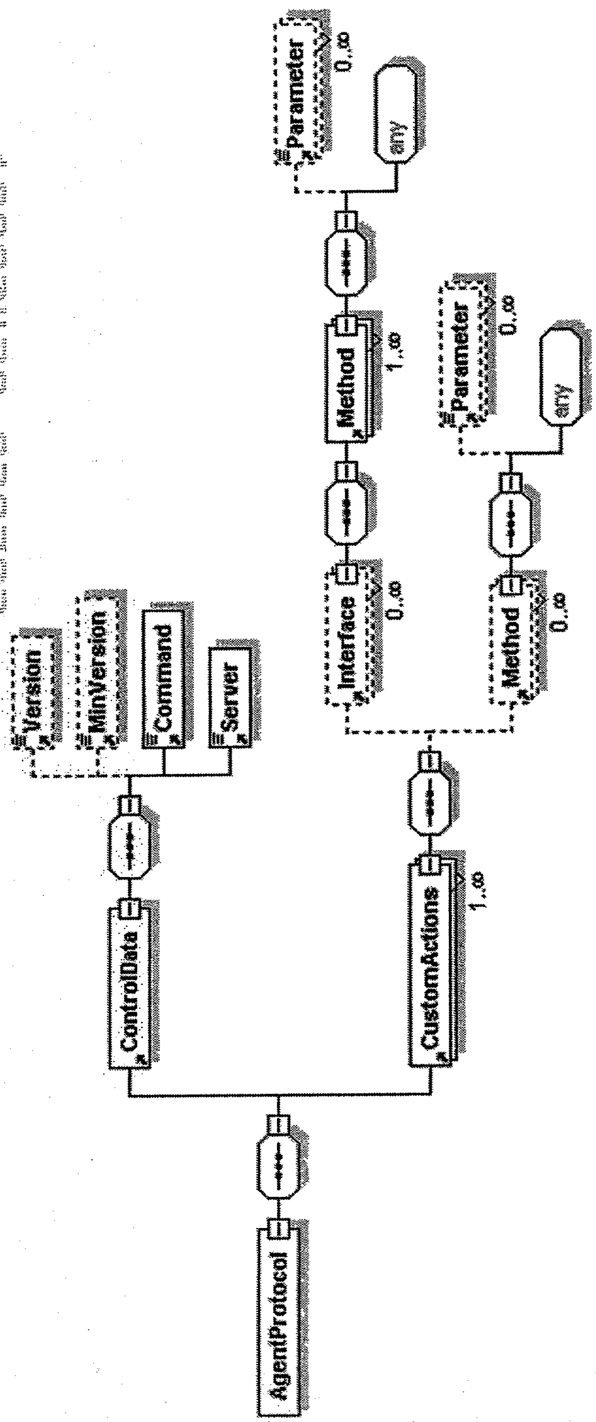
Fig. 7

Protocol Specification

<ControlData>			contains Agent specific control data like version information, the command to execute and computer information sending the custom actions information.
	<Command>		the command field could be used to easily determine if data is going to be retrieved (RequestCustomAction) or returned after the action has been executed (RespondToCustomAction) - optional.
	<Server>		computer information of the sender. If the Agent is able to process requests in parallel and asynchronosly, this information is useful. The simplest form of this field contains the computer name.
<CustomActions>			any number of custom action COM servers or dlls or executeables required for the tasks are listed within the CustomActions.
	id		the id specifies a CLSID if the custom action method is contained in a COM server or the path to a dll/exe file if a library or an executeable implements the method to execute. If a CLSID is specified, the following <Interface> complex type is required to specify the Interface of the COM server containing the method to execute. Otherwise <Interface> is not required and any number of <Method> types follow immediately.
	<Interface>		only required if the <Method> is implemented in a COM server.
	id		the Interface identifier of the COM server (IID).
		<Method>	any number of Methods implemented in the custom action device.
		id	the Method name. In case of a COM server, this is the method name of the COM interface, else this denotes the name of an exported function or e.g. a command line parameter of an executeable.
		<Parameter>	any number of parameters required for the method. This includes requested out parameters, which are listed, but don't contain data and inout parameters which have a different value on response than on request.
		id	the name of the parameter.
		type	can be any standard XML datatype.
		inout	possible values are 'in', 'out' and 'inout'. Specifies if the parameter is requested, passed to the function or passed to the function for modification.
		<any>	any other non standard XML datatype can follow.

Fig. 8

Fig. 9



CustomActionsProtocol.xsd

CustomActionsProtocolReq.xml


```
<?xml version="1.0" ?>
- <AgentProtocol xmlns="http://www.nai.com"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.nai.com CustomActionsProtocol.xsd">
  - <ControlData>
      <Version>0x01000001</Version>
      <MinVersion>0x01000001</MinVersion>
      <Command>RequestCustomAction</Command>
      <Server>ned1wnts2ke</Server>
    </ControlData>
  - <CustomActions
      id="<AGENT_INSTALLED_DIR>\\CustomActionsLibrary\\CustAct1.dll">
    - <Method id="GetRegStringValue">
        <Parameter id="Key" type="xs:string"
          inout="in"><AGENT_INSTALLED_REGKEY></Parameter>
        <Parameter id="Valuename" type="xs:string"
          inout="in">AgentVersion</Parameter>
        <Parameter id="Result" type="xs:string" inout="out" />
      </Method>
    </CustomActions>
  - <CustomActions id="{06E0062A-5069-4793-ACED-F80BE1BBC4AF}">
    - <Interface id="{C9E1CC03-8007-412A-8F5D-532C57DF4482}">
      - <Method id="ExecuteSilentInstallation">
          <Parameter id="ProductName" type="xs:string"
            inout="in">TestInstallProduct</Parameter>
          <Parameter id="ProductVersion" type="xs:decimal"
            inout="in">0x01000001</Parameter>
          <Parameter id="Location" type="xs:string"
            inout="in">c:\InstallImages</Parameter>
          <Parameter id="Result" type="xs:string" inout="out" />
        </Method>
      </Interface>
    - <Interface id="{C9E1CC03-8007-412A-8F5D-532C57DF4482}">
      - <Method id="GetSystemDirectory">
          <Parameter id="Directory" type="xs:string" inout="out" />
          <Parameter id="Result" type="xs:decimal" inout="out" />
        </Method>
      </Interface>
    </CustomActions>
  - <CustomActions id="{06E0062B-5069-4793-ACED-F80BE1BBC4AF}">
    - <Interface id="{A000CC03-8007-412A-8F5D-532C57DF4482}">
      - <Method id="TriggerEvent">
          <Parameter id="EventID" type="xs:decimal"
            inout="in">1000</Parameter>
          <Parameter id="EventDescription" type="xs:decimal"
            inout="in">The event %EventID% has been triggered by %
            USERNAME% on computer %COMPUTERNAME%. The %
            FILENAME% file is infected with %VIRUSNAME%. This has
            been detected by engineversion %ENGINEVERSION%
            datversion %DATVERSION%.</Parameter>
          <Parameter id="COMPUTERNAME" type="xs:string"
            inout="in">sourcecomputer</Parameter>
          <Parameter id="USERNAME" type="xs:string"
            inout="in">sourceuser</Parameter>
          <Parameter id="FILENAME" type="xs:string"
            inout="in">kernel32.dll</Parameter>
          <Parameter id="VIRUSNAME" type="xs:string"
```


Fig. 10A

CustomActionsProtocolReq.xml

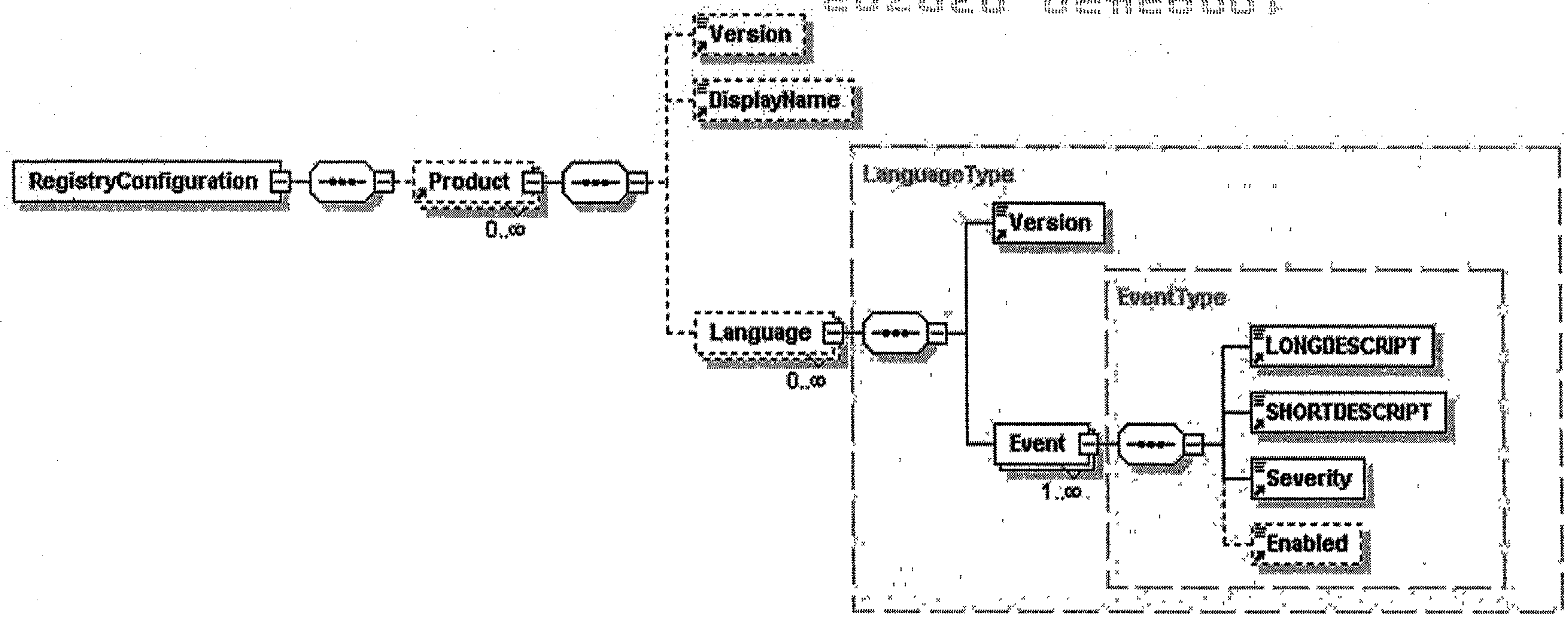
```
    inout="in">Nimbda</Parameter>
      <Parameter id="ENGINEVERSION" type="xs:decimal"
        inout="in">0x04005001</Parameter>
      <Parameter id="DATVERSION" type="xs:decimal"
        inout="in">0x07003009</Parameter>
      <Parameter id="Result" type="xs:string" inout="out" />
    </Method>
  </Interface>
 </CustomActions>
</AgentProtocol>
```


Fig. 10B

CustomActionsProtocolResp.xml

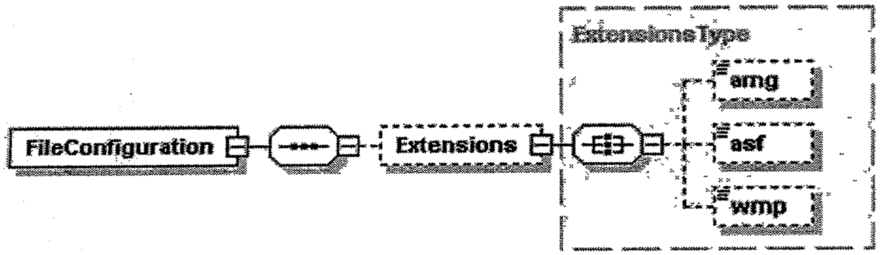

```
<?xml version="1.0" ?>
- <AgentProtocol xmlns="http://www.nai.com"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.nai.com CustomActionsProtocol.xsd">
  - <ControlData>
      <Version>0x01000001</Version>
      <MinVersion>0x01000001</MinVersion>
      <Command>RspondToCustomAction</Command>
      <Server>ned1wnts2ke</Server>
    </ControlData>
  - <CustomActions
      id="<AGENT_INSTALLED_DIR>\\CustomActionsLibrary\\CustAct1.dll">
    - <Method id="GetRegStringValue">
        <Parameter id="Result" type="xs:string"
          inout="out">5.0.1.10</Parameter>
      </Method>
    </CustomActions>
  - <CustomActions id="{06E0062A-5069-4793-ACED-F80BE1BBC4AF}">
    - <Interface id="{C9E1CC03-8007-412A-8F5D-532C57DF4482}">
      - <Method id="ExecuteSilentInstallation">
          <Parameter id="Result" type="xs:string" inout="out">Error: Invalid
            Image path specified.</Parameter>
        </Method>
      </Interface>
    - <Interface id="{C9E1CC03-8007-412A-8F5D-532C57DF4482}">
      - <Method id="GetSystemDirectory">
          <Parameter id="Directory" type="xs:string"
            inout="out">C:\Winnt\System32</Parameter>
          <Parameter id="Result" type="xs:decimal"
            inout="out">0</Parameter>
        </Method>
      </Interface>
    </CustomActions>
  - <CustomActions id="{06E0062B-5069-4793-ACED-F80BE1BBC4AF}">
    - <Interface id="{A000CC03-8007-412A-8F5D-532C57DF4482}">
      - <Method id="TriggerEvent">
          <Parameter id="Result" type="xs:string" inout="out">Event sent to
            testcomputer2</Parameter>
        </Method>
      </Interface>
    </CustomActions>
  </AgentProtocol>
```


Fig. 11



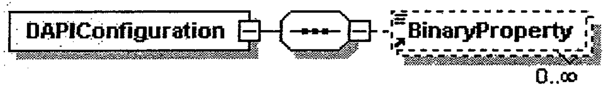
AgentConfiguration.xsd - RegistryConfiguration

Fig. 12



AgentConfiguration.xsd - FileConfiguration

Fig. 13



AgentConfiguration.xsd - DAPIConfiguration

Fig. 14

AgentConfigCustomAction.xml


```
<?xml version="1.0" ?>
- <AgentProtocol xmlns="http://www.nai.com"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.nai.com CustomActionsProtocol.xsd
   http://www.nai.com AgentConfiguration.xsd">
 - <ControlData>
     <Version>0x01000001</Version>
     <MinVersion>0x01000001</MinVersion>
     <Command>RequestCustomAction</Command>
     <Server>ned1wnts2ke</Server>
   </ControlData>
 - <CustomActions id="RegistryMapping.dll">
   - <Method id="WriteConfig">
     - <RegistryConfiguration
         id="HKEY_LOCAL_MACHINE\SOFTWARE\McAfee">
       - <Product id="Alert Manager">
           <Version>0x04070000</Version>
           <DisplayName>Alert Manager 4.7</DisplayName>
         - <Language id="0407">
             <Version>0x01000002</Version>
           - <Event id="1">
               <LONGDESCRIPT>Das ist eine Test-Nachricht von Alert
                 Manager.</LONGDESCRIPT>
               <SHORTDESCRIPT>Testing</SHORTDESCRIPT>
               <Severity>5</Severity>
               <Enabled>1</Enabled>
             </Event>
           </Language>
         - <Language id="0409">
             <Version>0x01000002</Version>
           - <Event id="1">
               <LONGDESCRIPT>This is an alert manager test
                 messge.</LONGDESCRIPT>
               <SHORTDESCRIPT>Testing</SHORTDESCRIPT>
               <Severity>0</Severity>
               <Enabled>1</Enabled>
             </Event>
           - <Event id="2">
               <LONGDESCRIPT>Text of event 2.</LONGDESCRIPT>
               <SHORTDESCRIPT>Testing</SHORTDESCRIPT>
               <Severity>1</Severity>
             </Event>
           </Language>
         </Product>
       </RegistryConfiguration>
     </Method>
   - <Method id="ReadConfig">
       <RegistryConfiguration
         id="HKEY_LOCAL_MACHINE\SOFTWARE\McAfee\*" />
     </Method>
   </CustomActions>
 - <CustomActions id="INIFileMapping.dll">
   - <Method id="WriteConfig">
     - <FileConfiguration id="C:\Program Files\Alert
         Manager\AMGConfig.ini">
       - <Extensions>
```

Fig. 15A

AgentConfigCustomAction.xml


```
        <amg>AMGConfig</amg>
        <asf>MPEGVideo</asf>
        <wmp>MPEGVideo2</wmp>
      </Extensions>
    </FileConfiguration>
  </Method>
- <Method id="ReadConfig">
    <FileConfiguration id="C:\Program Files\Alert
      Manager\AMGConfig.ini" />
  </Method>
</CustomActions>
- <CustomActions id="MAPIMapping.dll">
- <Method id="WriteConfig">
  - <DAPIConfiguration id="/O=org/OU=TestSite/CN=TestContainer">
      <BinaryProperty>0123456789ABCDEF00000</BinaryProperty>
    </DAPIConfiguration>
  </Method>
- <Method id="ReadConfig">
    <DAPIConfiguration id="/O=org/OU=TestSite/CN=TestContainer" />
  </Method>
</CustomActions>
</AgentProtocol>
```


Fig. 15B

Source

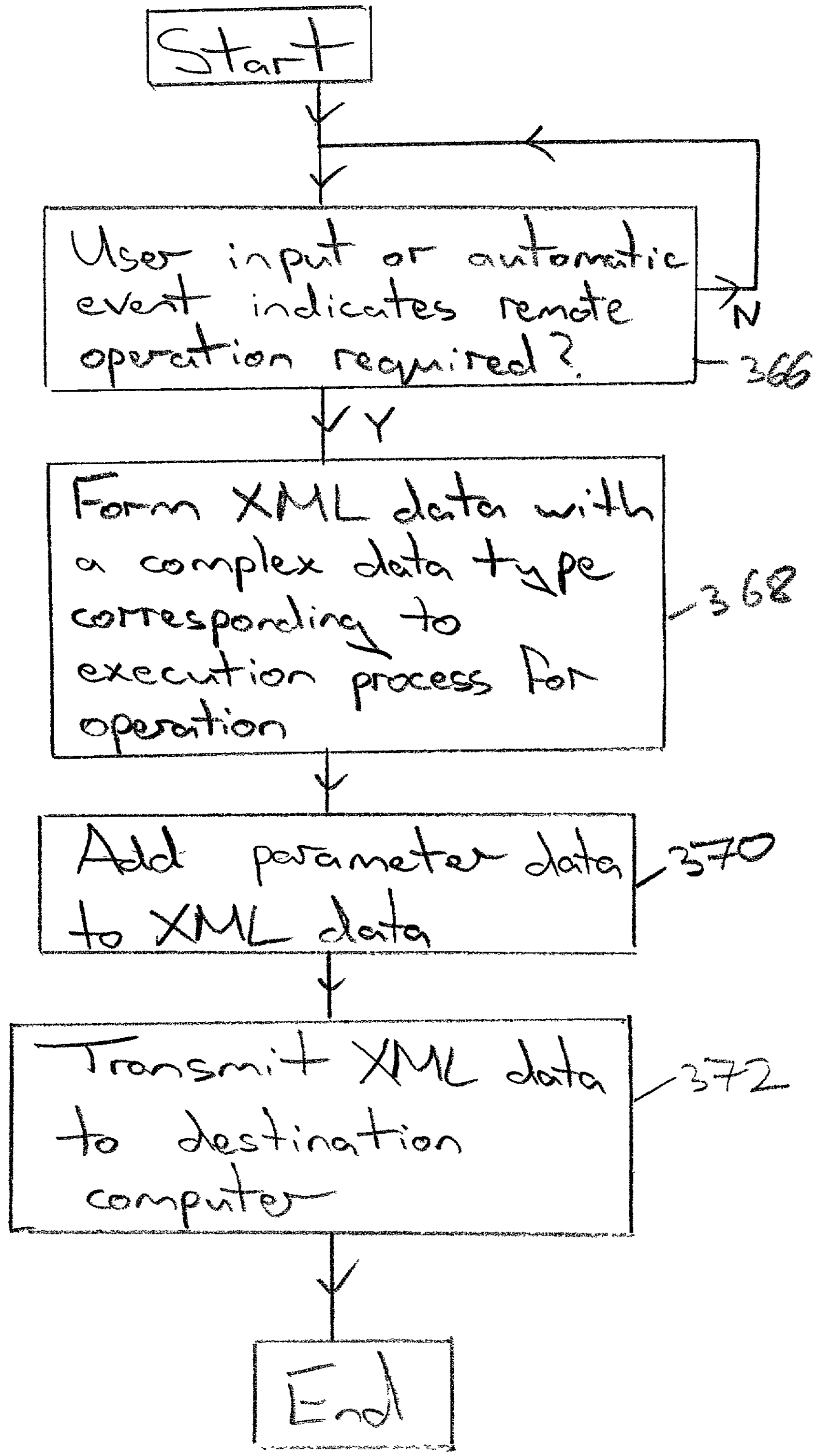


Fig. 16

Destination

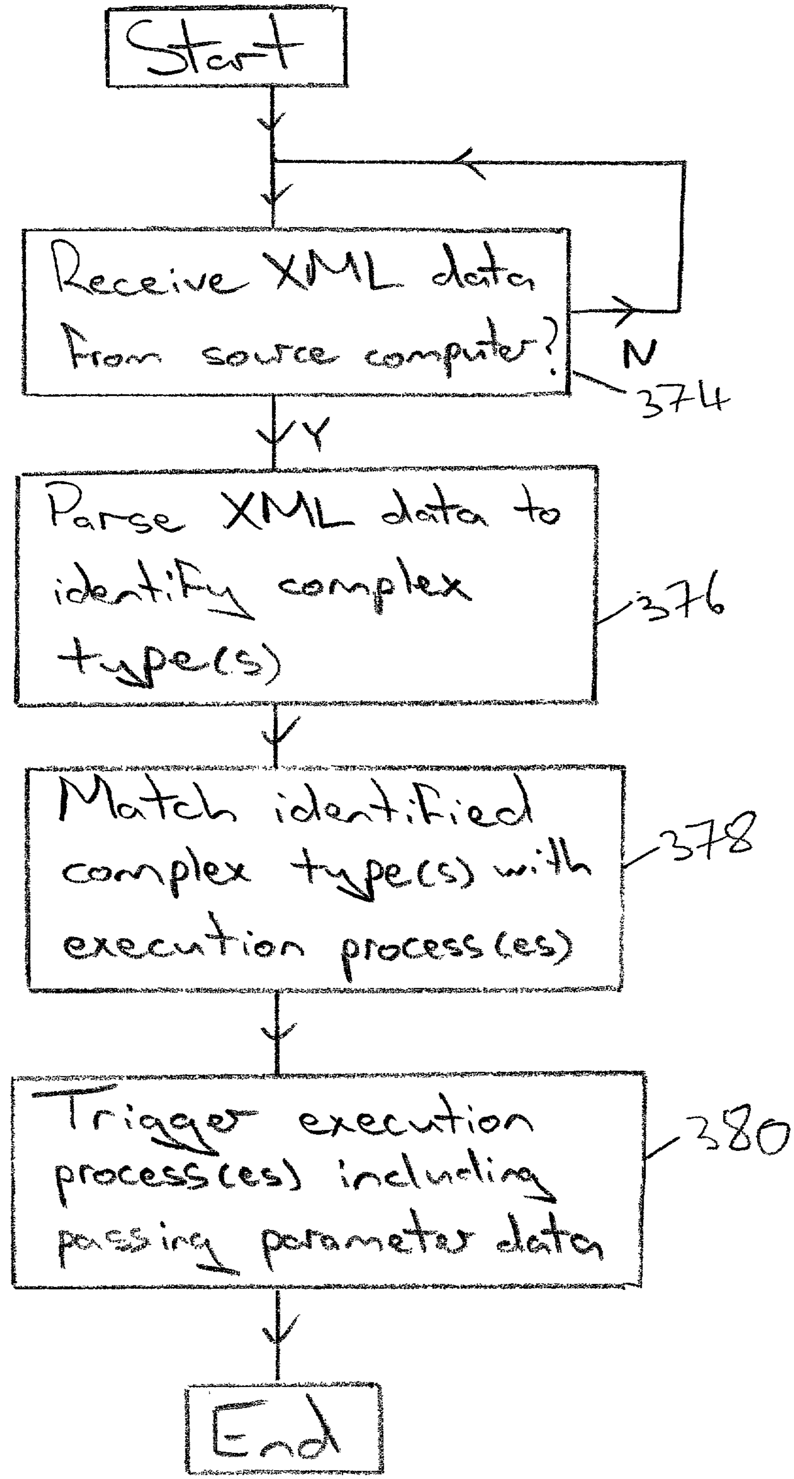
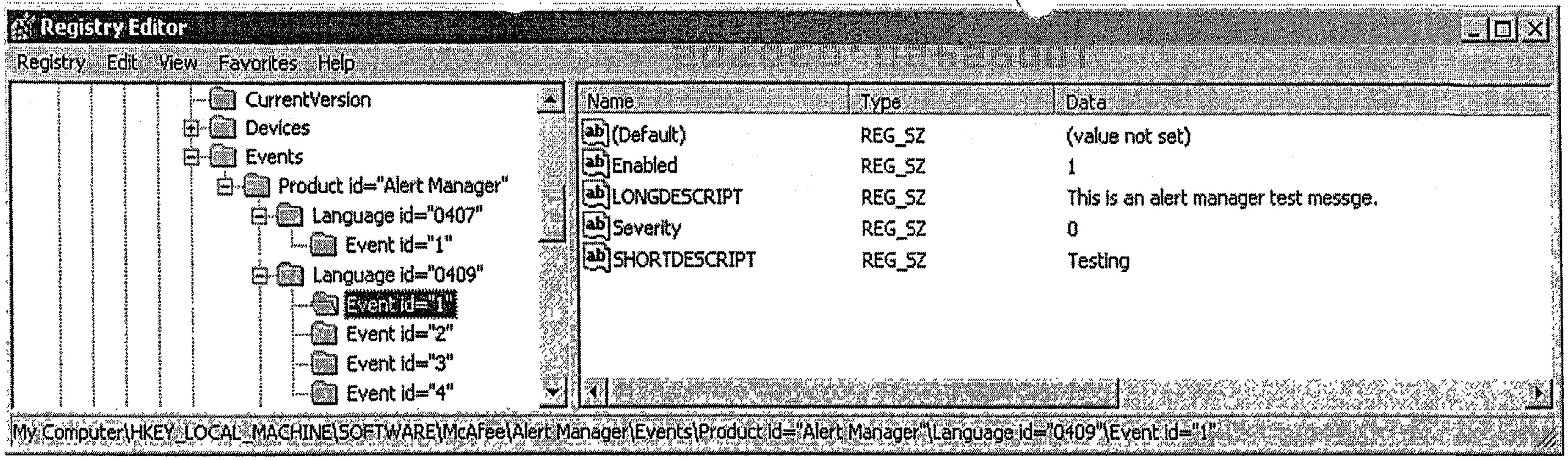
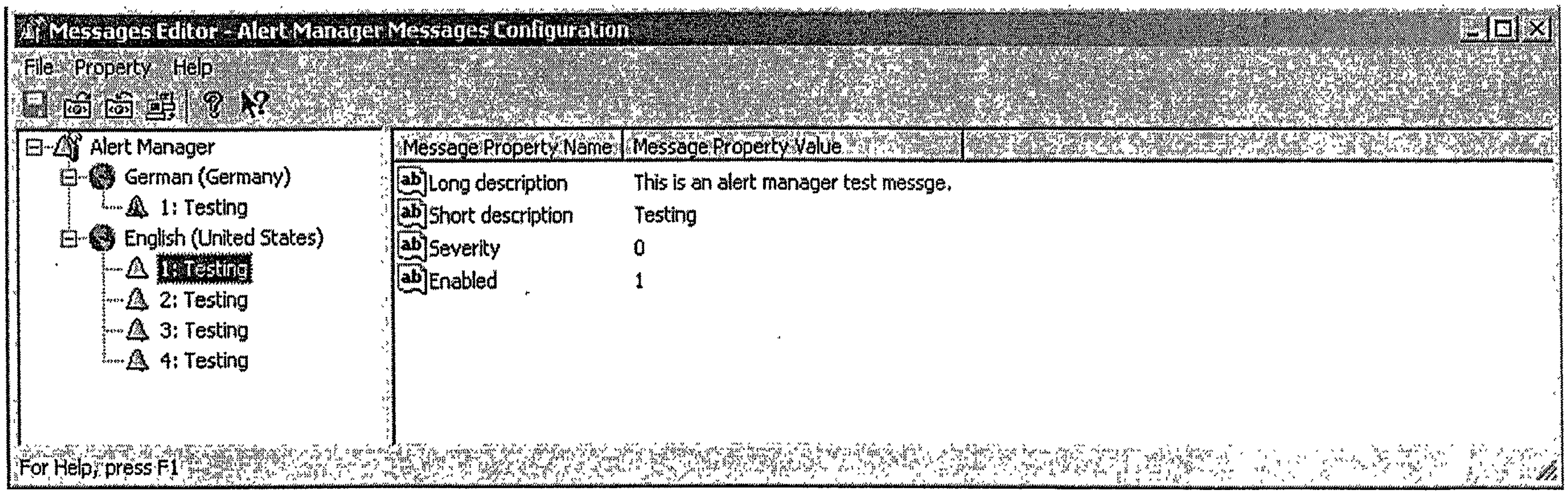


Fig. 17



Registry Data

Fig. 18



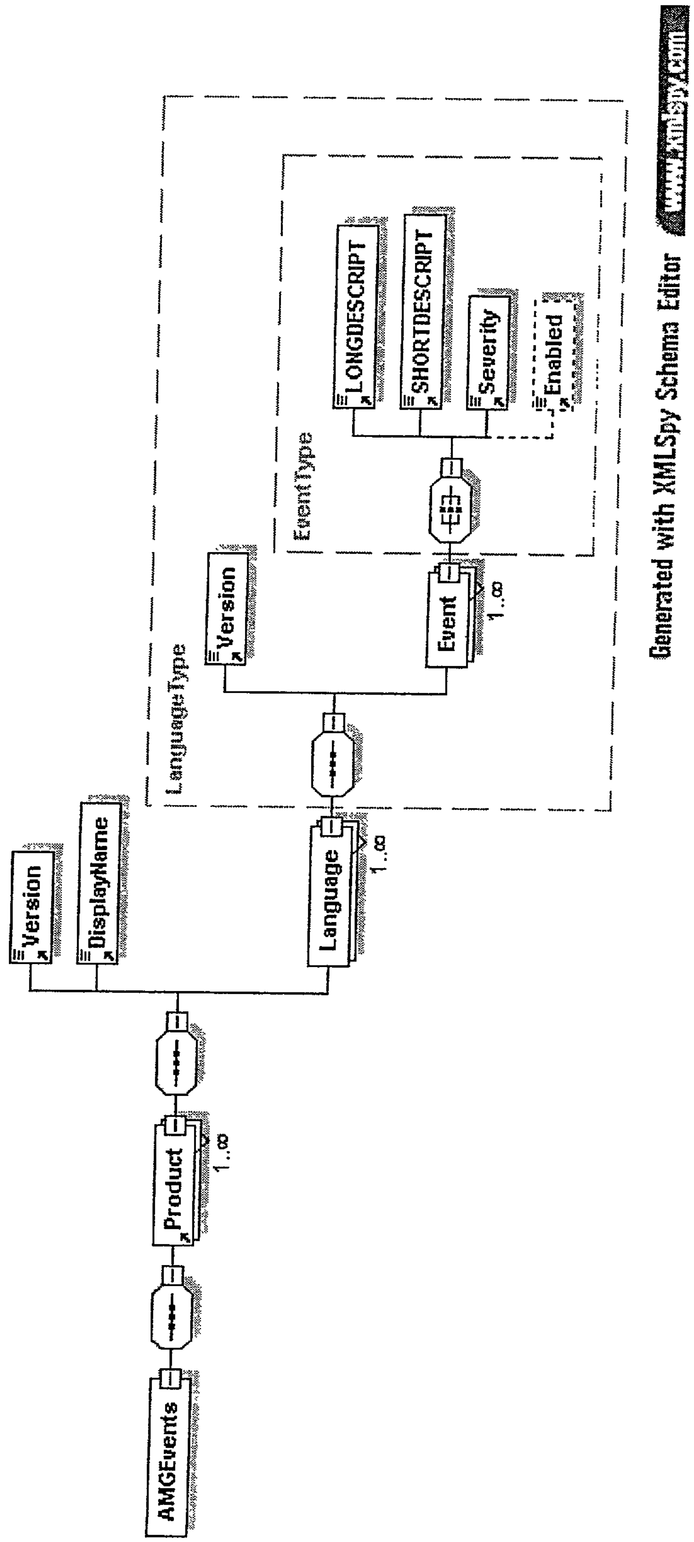
DOM Data View

Fig. 19

```
<?xml version="1.0" ?>
- <AMGEvents xmlns="http://www.nai.com"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.nai.com AMGEvents.xsd">
  - <Product id="Alert Manager">
      <Version>0x04070000</Version>
      <DisplayName>Alert Manager 4.7</DisplayName>
    - <Language id="0407">
        <Version>0x01000002</Version>
      - <Event id="1">
          <LONGDESCRIPT>Das ist eine Test-Nachricht von Alert
            Manager.</LONGDESCRIPT>
          <SHORTDESCRIPT>Testing</SHORTDESCRIPT>
          <Severity>5</Severity>
          <Enabled>1</Enabled>
        </Event>
      </Language>
    - <Language id="0409">
        <Version>0x01000002</Version>
      - <Event id="1">
          <LONGDESCRIPT>This is an alert manager test
            messge.</LONGDESCRIPT>
          <SHORTDESCRIPT>Testing</SHORTDESCRIPT>
          <Severity>0</Severity>
          <Enabled>1</Enabled>
        </Event>
      - <Event id="2">
          <LONGDESCRIPT>Text of event 2.</LONGDESCRIPT>
          <SHORTDESCRIPT>Testing</SHORTDESCRIPT>
          <Severity>1</Severity>
        </Event>
      - <Event id="3">
          <LONGDESCRIPT>Text of event 3.</LONGDESCRIPT>
          <SHORTDESCRIPT>Testing</SHORTDESCRIPT>
          <Severity>1</Severity>
        </Event>
      - <Event id="4">
          <LONGDESCRIPT>Text of event 4.</LONGDESCRIPT>
          <SHORTDESCRIPT>Testing</SHORTDESCRIPT>
          <Severity>1</Severity>
        </Event>
      </Language>
  </Product>
</AMGEvents>
```


XML Data

Fig. 20



XSD Data

Fig. 21

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- edited with XML Spy v4.0.1 U (http://www.xmlspy.com) by Napalm
(Napalm) -->
- <xs:schema targetNamespace="http://www.nai.com"
    xmlns="http://www.nai.com"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified">
    <xs:element name="DisplayName" type="xs:string" />
    <xs:element name="Enabled" type="xs:boolean" />
  - <xs:complexType name="EventType">
    - <xs:all>
        <xs:element ref="LONGDESCRIPT" />
        <xs:element ref="SHORTDESCRIPT" />
        <xs:element ref="Severity" />
        <xs:element ref="Enabled" minOccurs="0" />
      </xs:all>
      <xs:attribute name="id" type="xs:string" use="required" />
    </xs:complexType>
  - <xs:complexType name="LanguageType">
    - <xs:sequence>
        <xs:element ref="Version" />
        <xs:element name="Event" type="EventType"
          maxOccurs="unbounded" />
      </xs:sequence>
      <xs:attribute name="id" type="xs:string" use="required" />
    </xs:complexType>
  - <xs:element name="Product">
    - <xs:complexType>
      - <xs:sequence>
          <xs:element ref="Version" />
          <xs:element ref="DisplayName" />
          <xs:element name="Language" type="LanguageType"
            maxOccurs="unbounded" />
        </xs:sequence>
        <xs:attribute name="id" type="xs:string" use="required" />
      </xs:complexType>
    </xs:element>
  - <xs:element name="AMGEvents">
    - <xs:complexType>
      - <xs:sequence>
          <xs:element ref="Product" maxOccurs="unbounded" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="LONGDESCRIPT" type="xs:string" />
    <xs:element name="SHORTDESCRIPT" type="xs:string" />
    <xs:element name="Severity" type="xs:string" />
    <xs:element name="Version" type="xs:string" />
</xs:schema>
```


XSD Data

Fig. 22

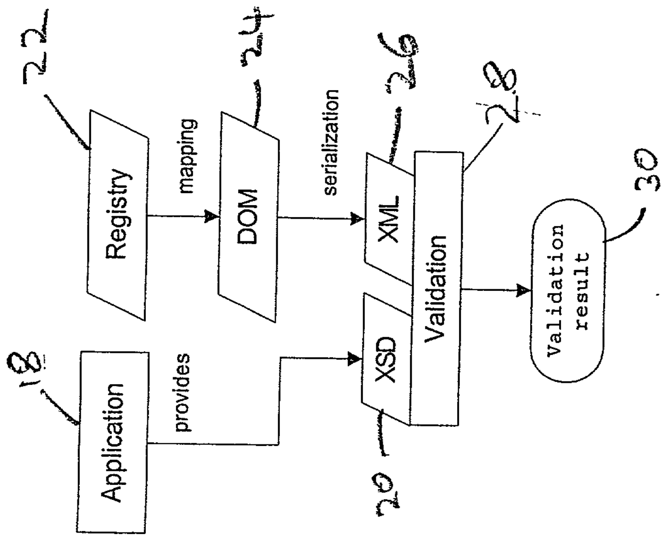

Application
18
provides
XSD
20
Validation
Registry
22
mapping
DOM
24
serialization
XML
26
28
Validation result
30

Fig. 24

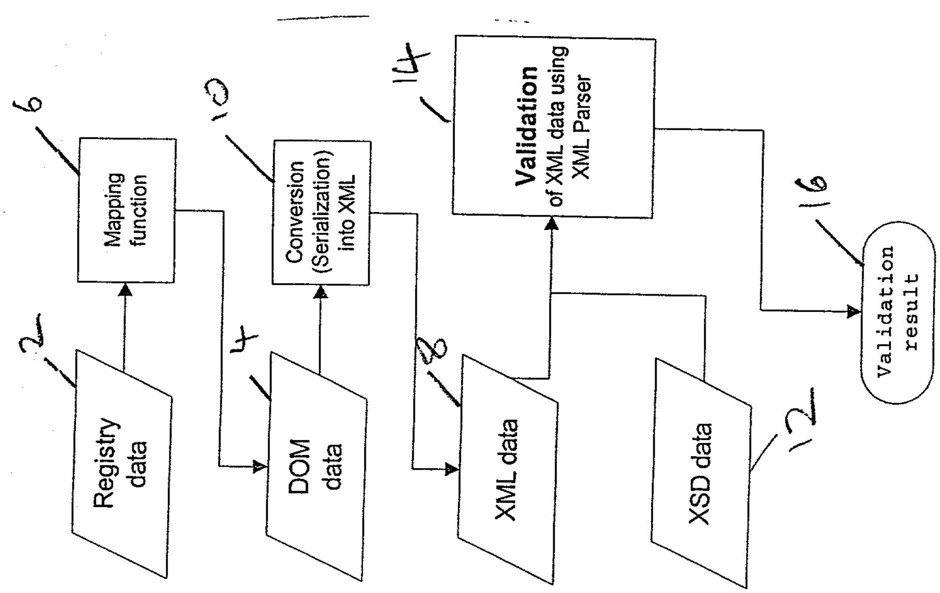

Registry data
2
Mapping function
6
DOM data
4
Conversion (Serialization) into XML
10
XML data
8
Validation of XML data using XML Parser
14
XSD data
12
Validation result
16

Fig. 23

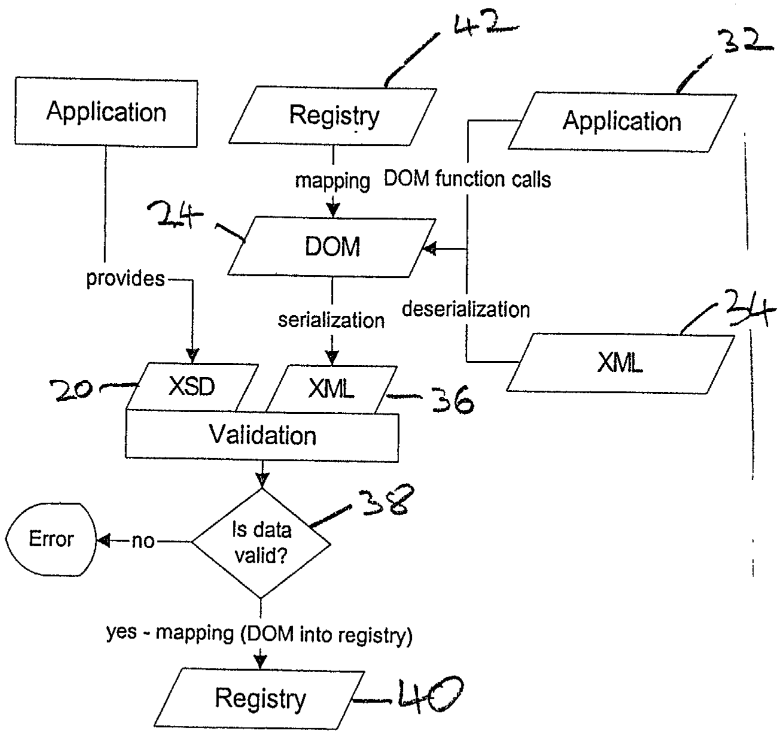

42
32
Application
Registry
Application
mapping
DOM function calls
24
DOM
provides
serialization
deserialization
34
XML
20
XSD
XML
36
Validation
38
Error
no
Is data valid?
yes - mapping (DOM into registry)
Registry
40

Fig. 25

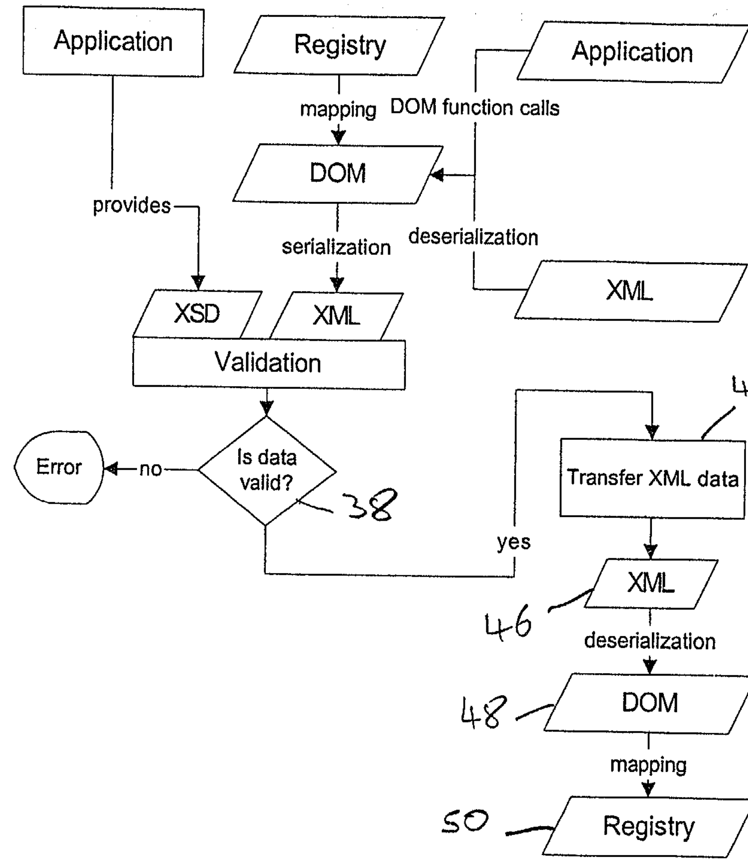

Application
Registry
Application
mapping
DOM function calls
DOM
provides
serialization
deserialization
XML
XSD
XML
Validation
44
Error
no
Is data valid?
38
Transfer XML data
yes
XML
46
deserialization
48
DOM
mapping
50
Registry

Fig. 26

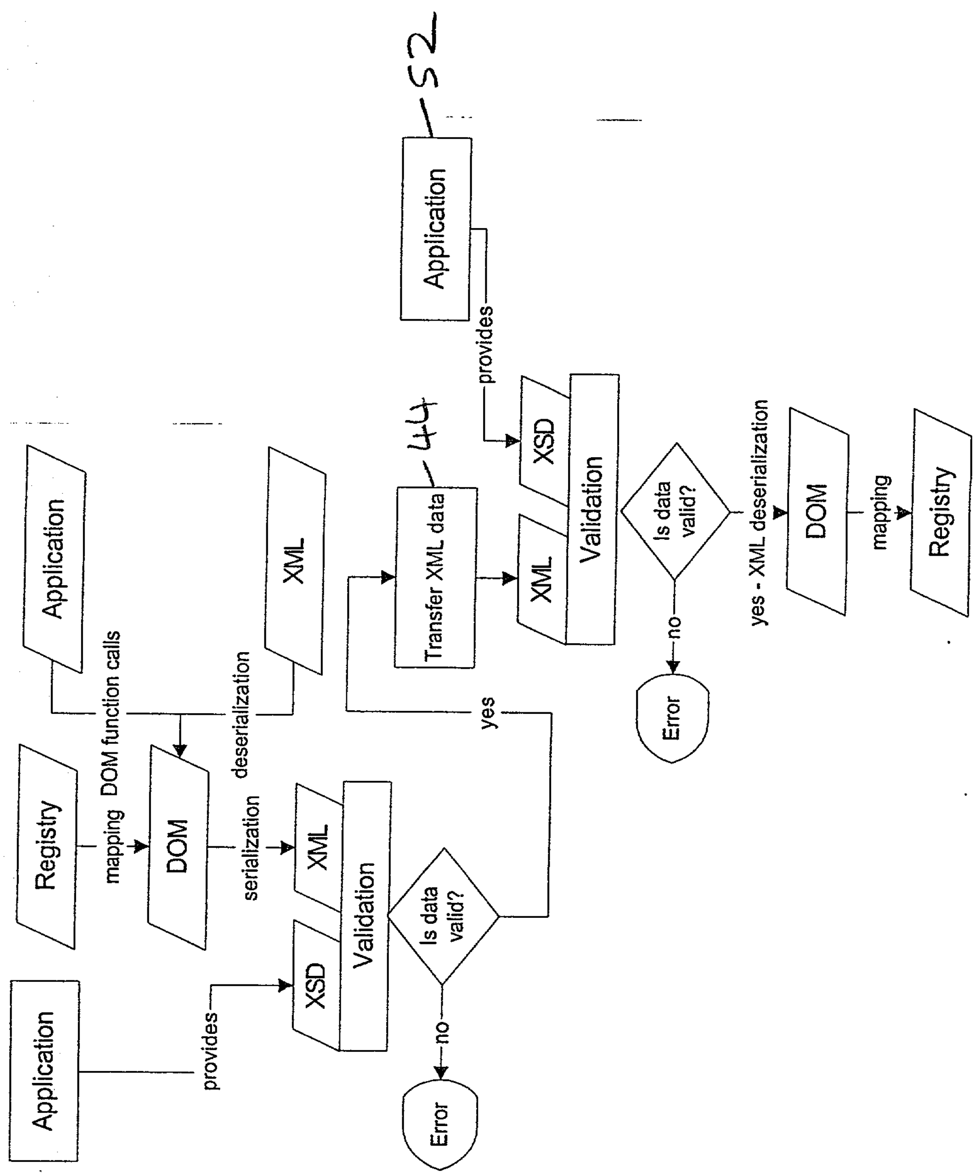

Application
Registry
mapping
DOM function calls
DOM
deserialization
XML
serialization
Application
provides
XSD
XML
Validation
Is data valid?
no
Error
yes
Transfer XML data
44
Application
52
provides
XSD
XML
Validation
Is data valid?
no
Error
yes - XML deserialization
DOM
mapping
Registry

Fig. 27

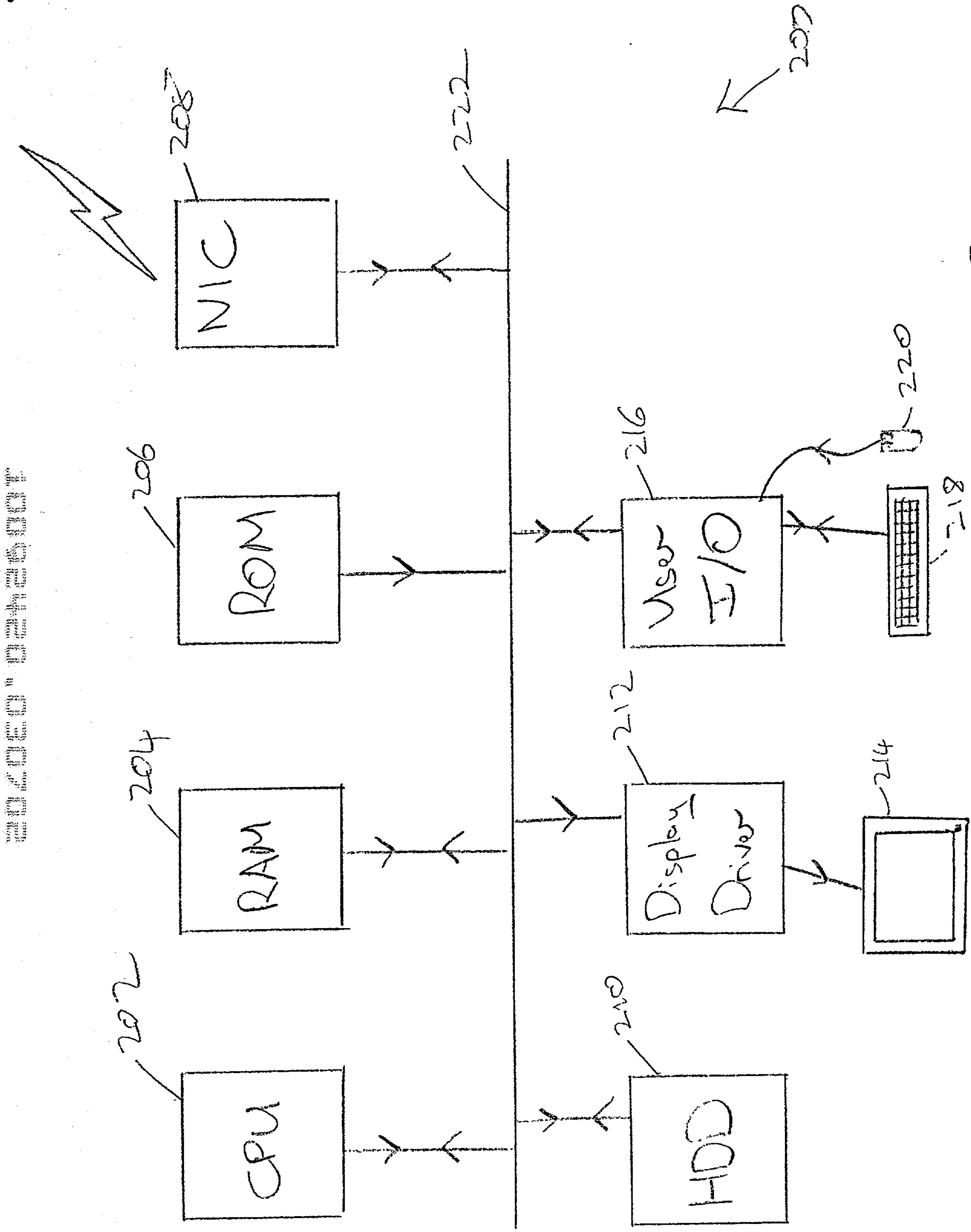

200
208
NIC
222
206
ROM
204
RAM
202
CPU
216
User I/O
220
218
212
Display Driver
214
210
HDD

Fig. 28